목 차

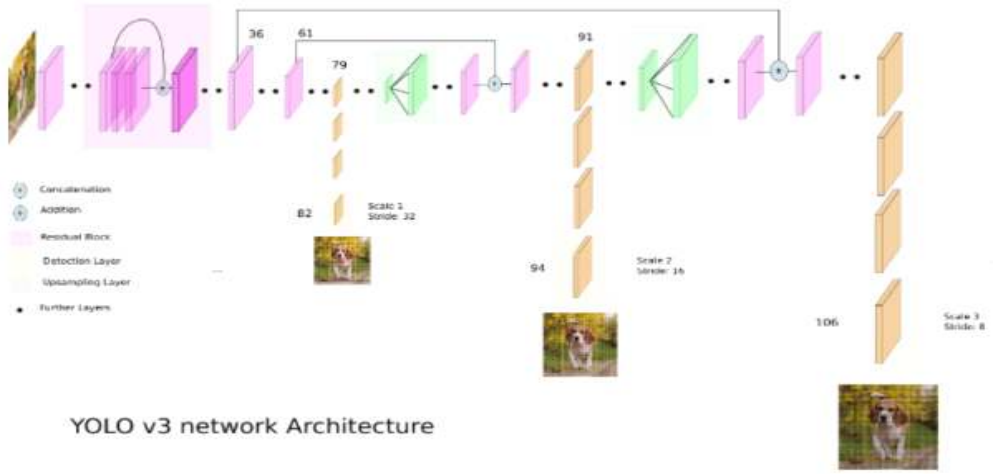
1. YOLO

1.1 YOLO란

YOLO는 'You Only Look Once'의 줄임말로, 객체 인식(Object Detection)의 대표적인 모델이다. 객체 인식이란 이미지 또는 비디오에서 개체를 식별하고 찾는 것과 관련된 컴퓨터 비전 작업이다. 특정 이미지에서 대상이 '무엇'인지, 이미지 내에서 객체의 '정확한 위치'는 어디인지 찾기 위한 작업이라 할 수 있다. YOLO는 입력된 이미지를 일정 분할로 그리드한 다음, 신경망을 통과하여 바운딩 박스와 클래스 예측을 생성해 최종 감지 출력을 결정한다. 실제 이미지를 테스트하기 전에 전체 데이터 세트에 대해 학습한다. 즉, YOLO는 이미지를 한 번 보는 것으로 객체의 종류와 위치을 추측하는 1-Stage 모델이라는 것이다.

YOLO는 Bbox(바운딩 박스)를 계산하기 위해 IoU(Intersect over Union) 및 NMS(Non-maximum suppression)의 주요 후처리 단계를 구현한다. IoU란 모델이 예측한 Bbox와 실체 객체의 Bbox가 얼마나 잘 일치하는지 확인하는 것이다. 실체로 이미지를 식별할 때, 특정 객체를 과도하게 식별하는 문제가 종종 발생한다. 이러한 문제는 실제 위치 근처에 여러개의 감지 그룹이 생성되는 방식으로 발생하는데, 감지 알고리즘이 아직 불완전하기 때문에 나타난다. 이런 현상을 방지하기 위해 NMS를 사용한다. NMS는 이미지에 객체라 판단한 Bbox 후보들 중 최적의 셀을 식별한다. 예를 들어, 이미지에 자전거라 판단되는 객체가 여러개 있다고 판단하는 것이 아닌, 동일한 객체에 대한 Bbox 중 가장 높은 확률을 가진 Bbox를 선택하는 것이다.

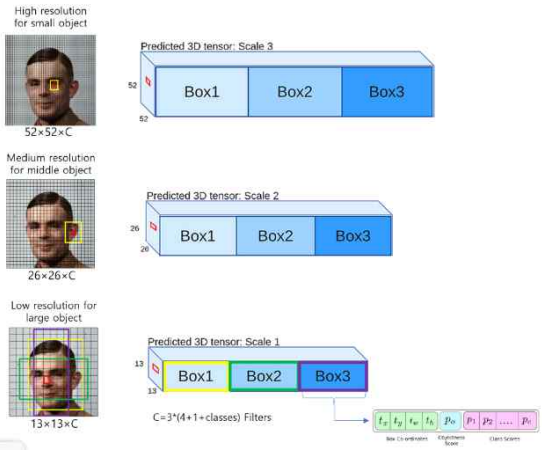
위와 같은 알고리즘을 통해 YOLO가 작동한다. 훈련을 받는 과정에서 전체 이미지를 확인한 뒤, 클래스에 대한 정보나 모양을 암시적으로 인코딩한다. 이후 객체 인식 여부를 테스트하기 위해 이미지를 입력했을 때, 해당 이미지를 N*N 그리드로 나누고 각 그리드마다 이미지 분류 및 지역화 작업을 한다. 객체가 이미지의 어느 위치에 있는지 확인하고, 식별해야 하는 객체에 Bbox를 그린다. 마지막으로, Bbox와 각 객체의 클래스 확률을 통해 객체를 인식하고 예측한다.



YOLO v3 network Architecture

YOLO의 장점은 처리 과정이 간단해 속도가 매우 빠르다는 것이다. 기존의 다른 real-time detection 모델과 비교할 때 2배 정도 높은 mAP를 보인다. 또, 이미지 전체를 한 번에 바라보는 방식으로 클래스에 대한 맥락적 이해도 역시 높다. 이로 인해 낮은 backgroud error(False-Positive)를 보인다. 또다른 장점으로는 객체에 대한 좀 더 일반화된 특징을 학습한다는 점이 있다. natural image로 학습하고 이를 artwork에 테스트 했을 때, 다른 Detection 모델들에 비해 훨씬 높은 성능을 보여준다. R-CNN 방식은 이미지 안에 객체가 있을만한 부분을 미리 예측하고, 컨볼루션 넷을 이용해 특징을 추출하는 과정을 거쳐야 하지만, YOLO는 이 단계를 간략화해 실시간 처리를 가능하게 했다.

반면, 몇 가지 한계점도 존재한다. 우선, 각 Grid Cell마다 8개의 Bbox만을 추측해야 한다는 공간적 제약성이 있어 가까이 붙어있는 물제와 작은 객체에 대해 상대적으로 낮은 정확도를 가진다. 또, 여러 개의 Down Sampling을 사용하기 때문에 디테일하지 못한 Feature가 드러날수도 있고, Localization이 부정확할수도 있다. 무엇보다 데이터로부터 Bbox를 훈련시키기 때문에, 학습 데이터에 없던 것이 시험 데이터로 주어질 경우 검출이 어렵다는 단점이 있다.



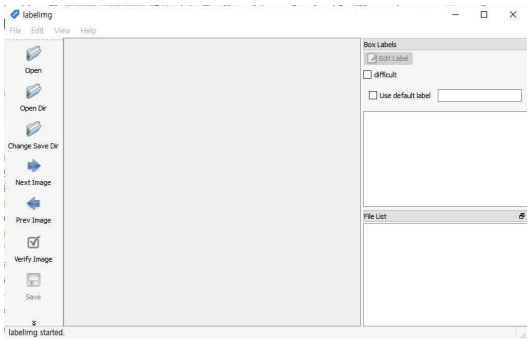
YOLO는 현재까지 v8까지 발전하였다. 이 중, 2018년에 발표된 YOLO v3 모델을 선정하였다. YOLO v3 모델은 v2에 비해 네트워크 구조와 학습 방법을 개선하여 객체 인식의 정확도와 속도를 대폭 개선한 모델이다. YOLO v3은 Backbone으로 Darknet 53을 사용하며, 그리드 당 비율과 크기가 다른 9개의 Anchor Box를 사용해 여러 Scale에서 나온 값을 사용한다. 작은 객체도 탐지가 가능하다는 것이다. 또, 어떤 객체가 사람이면서 남자, 또는 사람이면서 여자일 수도 있는 것처럼 한 물체에 대해 multi classlabel을 가질 수 있는데, softmax는 이렇게 한 box 안에 여러 class가 존재할 경우 적절하게 포착할 수 없었다. YOLO v3은 이런 점을 개선해 마지막에 Loss function을 하는 단계에서 softmax가 아닌 모든 class에 대해 sigmoid를 취해 각 class별로 binary classification을 하는 Independent logistic clssifier 방식을 적용하였다. v2에 비해 발전된 버전이라는 점과 이후 출시된 버전에 비해 비교적 구현이 어렵지 않다는 점을 이유로 YOLO v3 모델을 선정하였다.

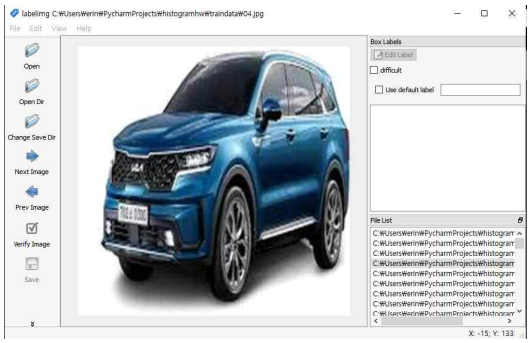
1.2 데이터 라벨링

가장 먼저, 사용할 데이터에 대한 라벨링 작업을 하였다. 클래스는 car, person, dog으로 총 3개를 선정하였으며, 각 클래스 당 25장의 이미지를 구글에 검색해 다운로드하였다. 이후, '딥러닝 yolov3 학습파일 만들기'의 영상을 참고하여 데이터 라벨링을 진행하였다. 라벨링 프로그램은 위 링크의 작성자가 공유한 'labelimg.exe' 실행파일을 사용하였다.

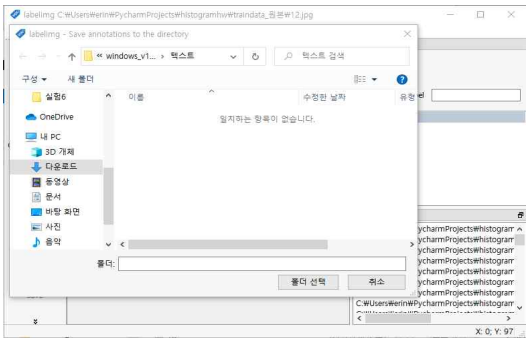
해당 프로그램을 사용하기에 앞서, 모든 이미지 파일을 jpg 파일로 변경하였다. jpg, jpeg, png 파일이 섞여 있으면 01.jpg, 01.jpeg, 01.png처럼 동일한 이름으로 저장했는데 알아채지 못할수도 있으며, 이후 코딩을 할 때 번거로울 수 있다.

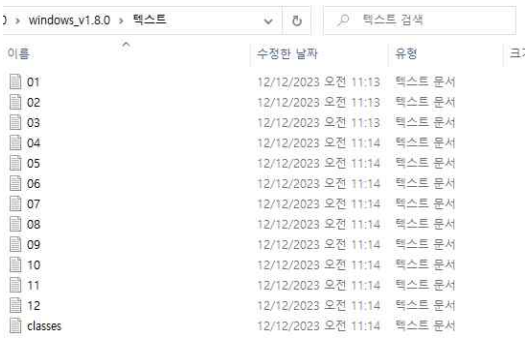
labelimg.exe를 실행하면 하단 왼쪽 그림과 같은 창이 실행된다. 이후 실행창의 상단에 있는 File을 클릭해, PascalVOC를 YOLO로 변경한다. 'Open Dir'을 선택해 이미지들을 모아놓았던 폴더를 선택하면 하단 오른쪽 그림과 같이 이미지 파일들이 빈 공간에 나타난다.



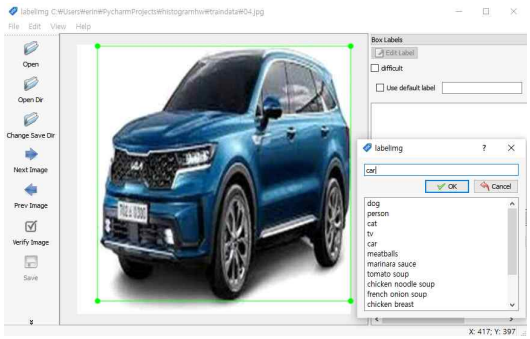


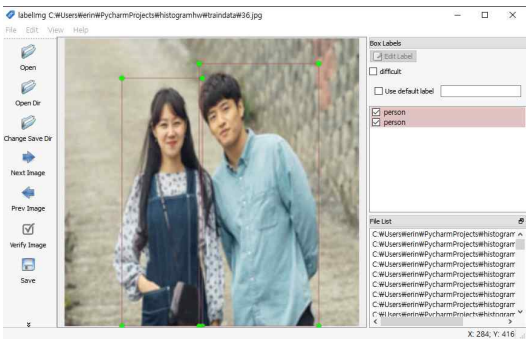
다음으로는, 라벨링한 데이터들이 저장된 폴더를 만들어야 한다. 원하는 위치에 '텍스트'라는 이름의 폴더를 만든 뒤, 폴더의 경로를 'labelimage.exe' 실행창의 'change save dir' 버튼을 클릭해 입력한다. 하단 왼쪽 그림처럼 폴더의 경로를 입력한 뒤, 폴더 선택을 클릭하면 된다. 이후 데이터 라벨링을 하면 하단 오른쪽 그림처럼 (클래스의 인덱스, box 중심의 x좌표, box 중심의 y좌표, box의 가로 길이, box의 세로 길이) 가 txt 파일로 저장된다. 가장 하단 classes.txt 파일에는 클래스명이 저장되어 각 인덱스가 어떤 클래스인지 알 수 있다.





위의 txt파일을 만들기 위해서는 다음 작업을 반복해야한다. 우선, 사각형을 그릴 수 있는 'Create ReacBox' 기능의 단축키인 w키를 누르고 객체의 Box를 그린다. 객체를 지정하면 하단 왼쪽 그림처럼 Box가 초록색 선으로 표시되며, 라벨링을 할 수 있는 창이 등장한다. 해당 창에 지정한 클래스명 (ex.car, person, dog)을 입력하면 된다. 이후 저장의 단축키인 ctrl + s로 저장하면 txt파일이 생성되는 것이다. 이 과정을 75장에 대해 반복하면 된다.





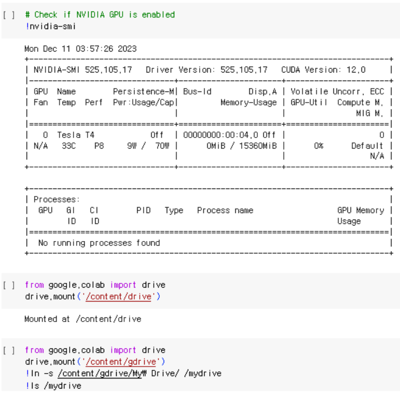
1.3 cfg 파일 및 기타 파일 준비

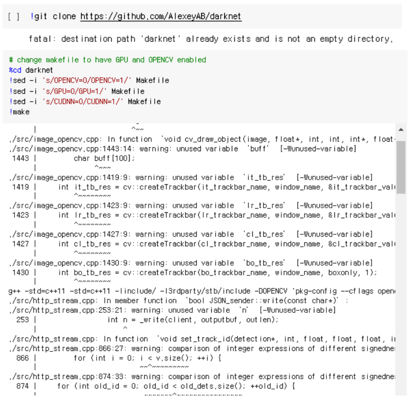
YOLO v3 방식으로 학습시키기 위해서는 cfg파일이 필요하다. 구글링을 통해 yolov3.cfg 파일을 다운로드한 뒤, 나의 방식에 맞게 내용을 수정하였다. 우선, max batches 값은 (클래스*2000)에 여유값 200을 더하므로, 3*2000+200=6200으로 수정하였다. steps 값도 max bathes 값의 (클래스*2000) 값의 10%와 20% 뺀 값이기 때문에 각각 6200-600=5600, 6200-1200=5000 값으로 수정하였다. 이후 ctrl + f로 yolo를 검색해 classes값은 3으로, filter 값은 (클래스+5)*3=(3+5)*3=24 값으로 수정하였다. yolov3.cfg 파일에는 yolo의 검색 결과가 3번이었기에 classes 값과 filter 값은 총 3번씩 수정해주었다.

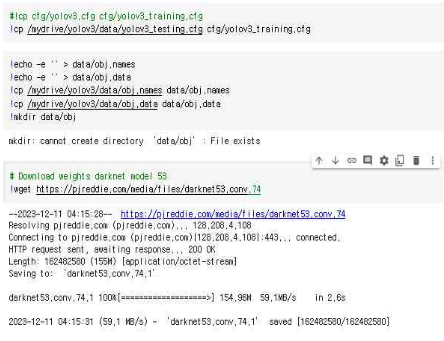
기타 필요했던 obj.data에서는 classes 값을 3으로, train.txt, val.txt, backup 등의 경로를 원하는 위치로 수정해주었고, 시도한 방법에 따른 파일들도 다운로드해 지정된 위치에 추가하였다.

1.4 데이터 학습 및 시험 방법 ①

‘딥러닝 yolov3 학습파일 만들기’와 ‘C# yolov3 사용하기(기초편)’의 내용을 참고해 라벨링한 데이터들을 학습시킨 뒤, 작동을 시험해보고자 하였다. 우선, weight 값을 얻기 위해 구글 코랩에서 학습을 시켜보았다. 구글 코랩의 GPU를 연결한 뒤, 코드를 실행해 GPU가 제대로 연결되었는지 확인하였다. 구글 드라이브와 구글 코랩도 연결한 뒤, 파일의 경로를 조회해 파일이 존재하는지 확인하였다. 하단 왼쪽 코드 사진을 보면, GPU와 드라이브가 모두 연결이 되어있다는 것을 확인할 수 있다.







YOLO v3은 Darknet을 기반으로 한 모델이기 때문에 Darknet의 클론을 설치하고 GPU를 사용하도록 컴파일하였다. 상단 중앙과 오른쪽 코드 사진을 보면 설치 현황을 확인할 수 있다. obj.names, obj.data 파일을 만들고 내용을 비우고, 경로를 복사하는 등의 코드도 실행하였다. draknet53.conv.74 라는 사전 훈련된 darknet 가중치를 다운로드한다. 이 가중치는 데이터를 학습시킬 때 초기 가중치로 사용될 예정이다.

```
!unzip /mydrive/yolov3/images.zip -d data/obj
  inflating: data/obj/15.jpg
 extracting: data/obj/15.txt
  inflating: data/obj/16.jpg
 extracting: data/obj/16.txt
  inflating: data/obj/17.jpg
 extracting: data/obj/17.txt
  inflating: data/obj/18.jpg
 extracting: data/obj/18.txt
  inflating: data/obj/19.jpg
 extracting: data/obj/19.txt
  inflating: data/obj/20.jpg
 extracting: data/obj/20.txt
  inflating: data/obj/21.jpg
  inflating: data/obj/21.txt
```

```
import glob
images_list = glob.glob("data/obj/*.jpg")
#images_list = glob.glob("data/obj/*.png")
print(images_list)

['data/obj/55.jpg', 'data/obj/27.jpg', 'data/obj/72.jpg', 'data/obj/29.jpg', 'data/obj/09.jpg', '
```

```
#Create training.txt file
file = open("data/train.txt", "w")
file.write("\n".join(images_list))
file.close()
```

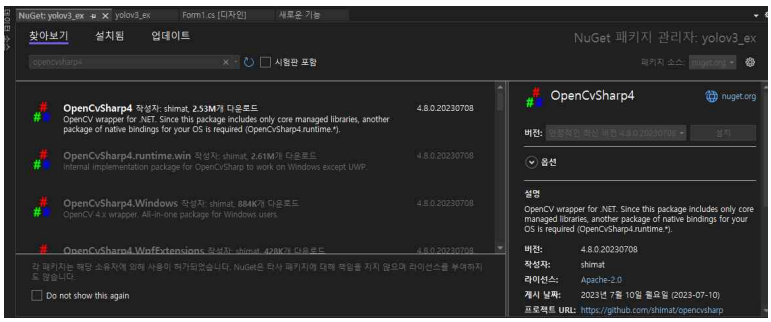
상단의 코드 사진들은 라벨링했던 이미지와 txt파일들을 zip파일로 압축해 드라이브에 업로드 했던 것을 불러와 unzip하는 과정이다. 이후, jpg 확장자를 가진 이미지 파일들을 검색해 리스트로 가져온다. 주석처리된 코드로 실행한다면 png 확장자를 가진 이미지 파일들을 가져오게 된다. 리스트는 image_list에 저장되는데, 저장된 이미지 파일들의 경로를 train.txt 파일에 한줄씩 입력하였다.

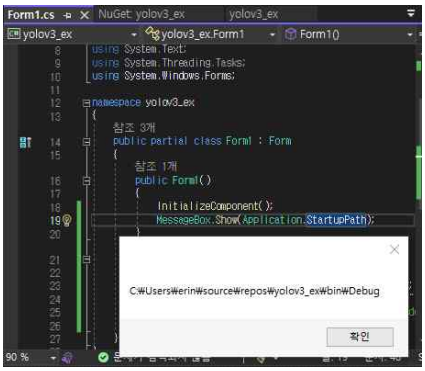
```
# Start the training
!./darknet detector train data/obj.data cfg/yolov3_training.cfg darknet53.conv.74 -dont_show
```

마지막으로 위 코드를 실행하면 학습이 시작된다. 클래스가 3개이기 때문에 cfg 파일 속 max batches 값을 6200으로 했었다. 그렇기에 1000 단위로 6200까지 weights 값이 기록된다.

- yolov3_training_1000.weights
- yolov3_training_2000.weights
- yolov3_training_3000.weights
- yolov3_training_4000.weights
- yolov3_training_5000.weights
- yolov3_training_6000.weights
- yolov3_training_final.weights
- yolov3_training_last.weights

다음으로는 얻은 weights 값을 이용해 학습이 제대로 되었는지 테스트하기 위해 'C# yolov3 사용하기 (기초편)'의 내용을 참고해 visual studio에서의 코딩을 시도해보았다. 'windows forms 앱(new framework)'로 새 프로젝트를 만든 뒤, 프레임 워크는 .NET 4.7.1로 설정하였다. OpenCV를 사용하기 위해 opencvsharp4를 설치하였다. 코드가 올바르게 작동한다면, 코드를 실행하면 디자인한 Form 창이 실행될 것이고, button을 누르면 테스트할 이미지 파일을 업로드할 수 있다. 이미지 파일을 업로드하면, 학습한 데이터들을 기반으로 이미지 속 객체에 Bounding box가 그려지고, 클래스명과 일치율도 표시가 될 것이다.

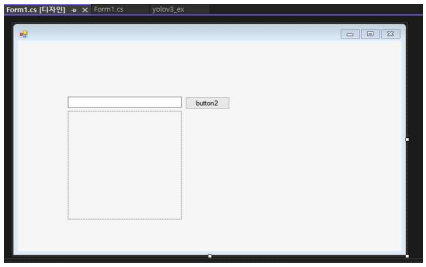





```
using OpenCvSharp.Dnn;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace yolov3_ex
{    public partial class Form1 : Form
    {         public Form1()
        {               InitializeComponent();
                        yolov3_Ready();
        }
        private void yolov3_Ready()
        {    // YOLOv3의 설정 파일과 모델 파일 경로 지정
            string cfg_Path = Application.StartupPath + @"\running\yolov3_testing.cfg";
            string model_Path = Application.StartupPath + @"\running\yolov3_training_last.weights";
            // OpenCVSharp.Dnn 네임스페이스의 CvDnn을 사용해 Darknet으로부터 모델 로드
            Net net = CvDnn.ReadNetFromDarknet(cfg_Path, model_Path);
            // 네트워크에 대한 기본적인 설정을 지정합니다.
            net.SetPreferableBackend((Backend)3); // 0(Defalt), 1(Halide), 2(Inference_engine), 3은 OpenCV 기반
            net.SetPreferableTarget((Target)2); // 0(CPU), 1(OpenCl), 2(OpenCl_FP16), 3(MyRiad), 4(FPGA)
        }
        string filename = "";
        // 이미지 파일을 선택하기 위한 버튼 클릭 이벤트 핸들러
        private void button2_Click(object sender, EventArgs e)
        {   OpenFileDialog ofd = new OpenFileDialog();
            if (ofd.ShowDialog() == DialogResult.OK)
        }
    }
}
```

상단의 코드까지 따라한 후, 실행을 하면 오른쪽 그림처럼 picture box, text box, button을 직접 추가한 창이 실행되고, button을 클릭해 이미지를 입력할 수 있어야 한다. 그러나 해당 코드를 실행해 보았을 때 아무런 오류가 없지만 Form 창이 실행되지 않았다. 여러 가지 시도를 해보고, 처음부터 다시 코드를 작성해보는 등 많은 시간을 투자해보았지만, 실행되지 않아 남은 코드만을 분석한 뒤, 다른 방법을 찾아보았다. 남은 코드는 아래와 같다.




```
private static void GetResult(Mat[] outs, bool nms, PictureBox pictureBox1, Image image, RichTextBox richTextBox1, bool morethreshold = true)
{    float threshold = 0.1f;//정확성 0~ 1
    float nmsThreshold = 0.1f;//상자 겹치는 거 필터작용
    try
   {        if (outs != null)
           {       List<int> classed = new List<int>();//해당 객체가 무엇인지
                   List<float> confidences = new List<float>();//전체 정확성
                   List<float> probilities = new List<float>();//정확성
                   List<Rect2d> boxes = new List<Rect2d>();//박스 그리기
                   int w = 0;  int h = 0;
                   if (image != null)
                   {     pictureBox1.Image = image;
                         w = image.Width;//이미지의 가로 길이 담기
                         h = image.Height;//이미지의 세로 길이 담기
                   }
                   else
                   {     w = pictureBox1.Image.Width;//이미지의 가로 길이 담기
                         h = pictureBox1.Image.Height;//이미지의 세로 길이 담기
                   }
                  const int prefix = 5;
                  richTextBox1.Text = "";
                  foreach (Mat mat in outs)
                  {       for (int i = 0; i < mat.Rows; i++)
                        {       float confidence = mat.At<float>(i, 4);//해당 객체의 정확성 담기
                                if (confidence > threshold)
                                {
              Cv2.MinMaxLoc(mat.Row(i).ColRange(prefix, mat.Cols), out _, out OpenCvSharp.Point max);
                                    int classes = max.X;//발견한 클래스의 넘버 값.
                                    float probability = mat.At<float>(i, classes + prefix);
                                    if (probability > threshold)
                                    {     float centerX = mat.At<float>(i, 0) * w;
                                          float centerY = mat.At<float>(i, 1) * h;
                                          float width = mat.At<float>(i, 2) * w;
                                          float height = mat.At<float>(i, 3) * h;
                                          classed.Add(classes);
                                          confidences.Add(confidence);
                                          probilities.Add(probability);
                                          boxes.Add(new Rect2d(centerX, centerY, width, height));
                                     }
                                  }
```

```
                    }
                }
                if (boxes.Count == 0)
                {   if (richTextBox1.Text != "관측되지 않음\n") richTextBox1.Text = "관측되지 않음\n";
                }
                else if (nms)
                {   CvDnn.NMSBoxes(boxes, confidences, threshold, nmsThreshold, out int[] indices);
                    foreach (int i in indices)
                    {   Rect2d box = boxes[i];
                        richTextBox1.AppendText($"{Label[classed[i]]} {probilities[i] * 100:0.0}%\n");
                        richTextBox1.AppendText("x,y = "+((int)box.X).ToString()+"/"+((int)box.Y).ToString()
                            +"\n"+"w,h="+((int)box.Width).ToString()+"/"+((int)box.Height).ToString()+"\n\n");
    Draw(classed[i], probilities[i], (float)box.X, (float)box.Y, (float)box.Width, (float)box.Height, pictureBox1);
                    }
                }
                else
                {   for (int i = 0; i < boxes.Count; i++)
                    {   Rect2d box = boxes[i];
                        richTextBox1.AppendText($"{ Label[classed[i]]} {probilities[i] * 100:0.0}% \n"); ;
                        richTextBox1.AppendText(((int)box.X).ToString() + "/" + ((int)box.Y).ToString() + "\n"
                                +((int)box.Width).ToString()+"/"+((int)box.Height).ToString()+"\n\n");
    Draw(classed[i], probilities[i], (float)box.X, (float)box.Y, (float)box.Width, (float)box.Height, pictureBox1);
                    }
                }
            }
        }
       catch
       {    richTextBox1.AppendText("GetResult 문에서 실패");
       }
}
private static void Draw(int classes, float probability, float centerX, float centerY, float width, float height,
PictureBox pictureBox1)  //Draw문
{   string text = $"{Label[classes]} ({probability * 100:0.0}%)";
    float x = centerX - width / 2;//중앙점에서 가로의 반을 빼줘야 그리는 점의 시작 포인트가 됨
    float y = centerY - height / 2;//중앙점에서 세로의 반을 빼줘야 그리는 점의 시작 포인트가 됨
    using (Graphics thumbnailGraphic = Graphics.FromImage(pictureBox1.Image))
    {   thumbnailGraphic.CompositingQuality = CompositingQuality.HighQuality;
        thumbnailGraphic.SmoothingMode = SmoothingMode.HighQuality;
        thumbnailGraphic.InterpolationMode = InterpolationMode.HighQualityBicubic;
        Font drawFont = new Font("Arial", 12, FontStyle.Bold);//폰트는 Arial 에 크기는 12, 굵기는 굵게
        SizeF size = thumbnailGraphic.MeasureString(text, drawFont);
        SolidBrush fontBrush = new SolidBrush(Color.Black);
        System.Drawing.Point atPoint = new System.Drawing.Point((int)x, (int)y - (int)size.Height - 1);;
        // Define BoundingBox options
        Pen pen = new Pen(color[classes], 3.2f);//객체마다 보여주는 색
        SolidBrush colorBrush = new SolidBrush(color[classes]);
       thumbnailGraphic.FillRectangle(colorBrush,(int)x,(int)(y-size.Height-1),(int)size.Width,(int)size.Height);
        thumbnailGraphic.DrawString(text, drawFont, fontBrush, atPoint);//사각형 그리기
        thumbnailGraphic.DrawRectangle(pen, x, y, width, height); //Bbox 그리기
    }
}
```

1.5 데이터 학습 및 시험 방법 ②

두 번째 방법은 학습부터 테스트까지 구글 코랩을 사용하는 방법을 찾아보았다. 구글 코랩에서 일정 시간 사용 가능하게 해주는 GPU를 이용하고, YOLO v3을 학습시키는데 필요한 darknet 라이브러리를 설치하였다. 이를 위한 코드는 하단 그림과 같다. 코드 캡처를 이후에 했기 때문에, 구글 코랩의 GPU 사용 시간이 경과해 코드가 실행되지 않은 것처럼 보인다. 하지만, 코드를 실행해보았을 때, 제대로 작동함을 확인하였다.

```
[42] # clone darknet repo
     !git clone https://github.com/AlexeyAB/darknet

     fatal: destination path 'darknet' already exists and is not an empty directory.
```

```
[4]  # change makefile to have GPU and OPENCV enabled
     %cd darknet
     !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
     !sed -i 's/GPU=0/GPU=1/' Makefile
     !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
     !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
     !sed -i 's/LIBSO=0/LIBSO=1/' Makefile


     # verify CUDA
     !/usr/local/cuda/bin/nvcc --version

     /content/darknet
     nvcc: NVIDIA (R) Cuda compiler driver
     Copyright (c) 2005-2022 NVIDIA Corporation
     Built on Wed_Sep_21_10:33:58_PDT_2022
     Cuda compilation tools, release 11.8, V11.8.89
     Build cuda_11.8.r11.8/compiler.31833905_0
```

```
[5]  # make darknet (builds darknet so that you can then use the darknet executable file to run or train object detectors)
     !make
     ./src/gaussian_yolo_layer.c:118:42: warning: passing argument 1 of 'cudaHostAlloc' from incompatible pointer type [-Wincompatible-pointer-types]
       118 |         if (cudaSuccess != cudaHostAlloc(&l->delta, l->batch*l->outputs * sizeof(float), cudaHostRegisterMapped)) {
           |                                          ^~~~~~~~~
           |                                          |
           |                                          float **
     In file included from /usr/local/cuda/include/cuda_runtime.h:96,
                      from include/darknet.h:41,
                      from ./src/gaussian_yolo_layer.h:5,
                      from ./src/gaussian_yolo_layer.c:7:
     /usr/local/cuda/include/cuda_runtime_api.h:5202:60: note: expected 'void **' but argument is of type 'float **'
      5202 | extern __host__ cudaError_t CUDARTAPI cudaHostAlloc(void **pHost, size_t size, unsigned int flags);
           |                                                     ~~~~~~~^~~~~
     gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/inclu
     gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/inclu
     gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/inclu
     gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/inclu
     gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/inclu
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencode arch=compute_61,code=[sm_61,compute_6
     g++ -std=c++11 -std=c++11 -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -
     g++ -std=c++11 -shared -std=c++11 -fvisibility=hidden -DLIB_EXPORTS -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/r
     In file included from src/yolo_v2_class.cpp:2:
     include/yolo_v2_class.hpp: In member function 'void track_kalman_t::clear_old_states()':
     include/yolo_v2_class.hpp:879:50: warning: comparison of integer expressions of different signedness: 'unsigned int' and 'int' [-Wsign-compare]
```

학습에 필요한 cfg 파일, obj.name 파일, 데이터 세트들은 앞선 방법에 사용했던 파일을 구글 드라이브에 그대로 사용하였다. 코드를 실행하면 데이터 세트를 train data와 valid data로 분류해 각각 txt 파일로 저장하고, 학습할 때 txt 파일의 내용을 참고해 학습 데이터와 검증 데이터를 분류할 것이다. 하단 코드 캡쳐본을 보면, 총 75장인 이미지 파일이 모두 리스트로 만들어졌으며, 그중 67개가 train data, 8개가 valid data로 분류된 것을 알 수 있다. 분류된 data들을 txt파일로 저장하는 것 역시 'Done'이 출력됨으로써 성공적으로 작동된 것을 확인할 수 있다.

```
[6]  # Connect the Colab notebook to Google Drive
     from google.colab import drive
     drive.mount('/content/gdrive')

     Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
[7]  # check your data
     %cd /
     from glob import glob
     dataset_dir = "/content/gdrive/MyDrive/dataset/"

     img_list = glob(dataset_dir+'*.jpg')
     # you should have images with labels.txt in same folder
     print("your images :",len(img_list))

     /
     your images : 75
```

```
from sklearn.model_selection import train_test_split
train_img_list, val_img_list = train_test_split(img_list, test_size=0.1, random_state=42)
print(len(train_img_list), len(val_img_list))

67 8
```

```
with open('/content/gdrive/MyDrive/data/train.txt', 'w') as f:
  f.write('₩n'.join(train_img_list) + '₩n')

with open('/content/gdrive/MyDrive/data/val.txt', 'w') as f:
  f.write('₩n'.join(val_img_list) + '₩n')

print("Done")

Done
```

학습에 필요한 파일들의 경로를 알려주는 obj.data 파일을 업로드한 뒤, darknet53 파일을 다운로드한 후, 학습을 시작한다. 이 때, obj.data 파일 속 backup의 경로에 weights 파일들이 저장된다. 학습할 데이터가 많고 올바르게 작동한 경우, last.weights와 best.weights 파일이 저장된다. 75장의 이미지 파일을 대상으로 시행했을 때는 best.weights 파일은 저장되지 않고, last.weights 파일만이 저장되었다. 하단의 사진들은 만들어진 last.weights 파일과 실행한 코드들이다.

yolov3_training_last.weights	나	오전 4:20	234.9MB	⋮

```
%cd /content/darknet/
!wget https://pjreddie.com/media/files/darknet53.conv.74
```

```
/content/darknet
--2023-12-11 11:46:17--  https://pjreddie.com/media/files/darknet53.conv.74
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 162482580 (155M) [application/octet-stream]
Saving to: 'darknet53.conv.74'

darknet53.conv.74   100%[===================>] 154.96M  21.0MB/s    in 8.4s

2023-12-11 11:46:27 (18.4 MB/s) - 'darknet53.conv.74' saved [162482580/162482580]
```

```
%cd /content/darknet
!./darknet detector train /content/gdrive/MyDrive/obj.data\
                          /content/gdrive/MyDrive/detect.cfg\
                          darknet53.conv.74\
                          -dont_show -map
```

```
Wrong annotation: class_id = 4. But class_id should be [from 0 to 2], file: /content/gdrive/MyDrive/dataset/14.txt

Wrong annotation: class_id = 4. But class_id should be [from 0 to 2], file: /content/gdrive/MyDrive/dataset/24.txt

Wrong annotation: class_id = 4. But class_id should be [from 0 to 2], file: /content/gdrive/MyDrive/dataset/23.txt

Wrong annotation: class_id = 4. But class_id should be [from 0 to 2], file: /content/gdrive/MyDrive/dataset/07.txt
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.544963), count: 6, class_loss = 0.683643, iou
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.000002, iou
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, ic
 total_bbox = 6360, rewritten_bbox = 0.000000 %


 Tensor Cores are disabled until the first 3000 iterations are reached.
 (next mAP calculation at 1000 iterations) 998/6200: loss=0.4 hours left=0.8
 998: 0.369277, 0.368266 avg loss, 0.000992 rate, 0.589738 seconds, 7984 images, 0.821781 hours left
Loaded: 0.000086 seconds
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.742101), count: 3, class_loss = 0.363749, iou
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.000001, iou
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, ic
 total_bbox = 6363, rewritten_bbox = 0.000000 %


 Tensor Cores are disabled until the first 3000 iterations are reached.
 (next mAP calculation at 1000 iterations) 999/6200: loss=0.2 hours left=0.8
 999: 0.183798, 0.349819 avg loss, 0.000996 rate, 0.583899 seconds, 7992 images, 0.822086 hours left
Loaded: 0.000087 seconds

Wrong annotation: class_id = 4. But class_id should be [from 0 to 2], file: /content/gdrive/MyDrive/dataset/02.txt

Wrong annotation: class_id = 4. But class_id should be [from 0 to 2], file: /content/gdrive/MyDrive/dataset/01.txt
```

```
[21] !./darknet detector map /content/gdrive/MyDrive/obj.data\
                             /content/gdrive/MyDrive/detect.cfg\
                             /content/gdrive/MyDrive/weights
```

```
   71 Shortcut Layer: 68,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
   72 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
   73 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
   74 Shortcut Layer: 71,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
   75 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
   76 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
   77 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
   78 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
   79 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
   80 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
   81 conv     24       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x  24 0.008 BF
   82 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.00
   83 route  79                                     ->   13 x  13 x 512
   84 conv    256       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x 256 0.044 BF
   85 upsample                 2x    13 x  13 x 256 ->   26 x  26 x 256
   86 route  85 61                                  ->   26 x  26 x 768
   87 conv    256       1 x 1/ 1     26 x  26 x 768 ->   26 x  26 x 256 0.266 BF
   88 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
   89 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
   90 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
   91 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
   92 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
   93 conv     24       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x  24 0.017 BF
   94 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.00
   95 route  91                                     ->   26 x  26 x 256
   96 conv    128       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x 128 0.044 BF
   97 upsample                 2x    26 x  26 x 128 ->   52 x  52 x 128
```

마지막으로, 테스트할 이미지를 구글 드라이브에 저장하고 제대로 학습이 되었는지 확인한다.

```
from IPython.display import clear_output
# define helper functions
def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
  %matplotlib inline

  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)

  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()
```

```
# run your custom detector with this command (upload an image to your google drive to test, thresh flag sets accuracy that detection must be in order to show it)
!./darknet detector test /content/gdrive/MyDrive/obj.data\
                         /content/gdrive/MyDrive/detect.cfg\
                         /content/gdrive/MyDrive/weights/yolov3_training_last.weights\
                         /content/gdrive/MyDrive/testimage/25.jpg
clear_output()
imShow('predictions.jpg')
```

상단 오른쪽 사진이 테스트 이미지를 업로드한 결과이다. 코드가 올바르게 작동하고, YOLO v3 모델이 데이터 세트들을 잘 학습하고, 올바르게 예측했다면, Bounding box가 그려지고 예측치가 표시되었을 것이다. 하지만, 30장의 테스트 파일을 업로드 해보았을 때, 아무것도 표시되지 않았다. YOLO v3 모델 구현에 실패한 것이라고 할 수 있다.

1.6 실패 사유 분석 및 고찰

YOLO v3 모델 구현에 실패한 사유는 다양하다고 생각한다. 우선, 학습시킬 데이터 세트의 양이 매우 적었다. 모델을 학습시키는데 클래스별로 25장씩 총 75장밖에 사용하지 않았지만, 모델이 제대로 작동하려면 최소 수백장부터 수천장까지 필요하다고 한다. 더욱이, 구글에서 검색한 이미지이기 때문에 객체의 각도, 포즈, 구도, 모양이 매우 다양했기에 더욱 학습이 어려웠던 것이라 판단한다. 배경이 화려한 이미지들도 다수 있었기 때문에 매우 적은 양의 데이터 세트로 학습하기에는 어려움이 있지 않았나 생각한다.

또 다른 이유로는 코딩 실력 부족이 있다. 텐서플로, 파이토치 등 딥러닝 모델 구현에 필요한 라이브러리들을 사용해본 적이 없고, 코드가 복잡해지면 해석할 수 없기 때문에 코드에 오류가 나도 해결이 어렵다. 챗GPT, 바드 등의 도움을 받고자 했지만, 코드 자체를 완벽히 이해하지 못한 상황이었기 때문에 오류를 수정할 수 없었다. 깃허브 역시 이전에 사용해본 적이 없기 때문에 파일을 다운받더라도 나의 목적에 맞게 코드를 수정하거나 데이터를 바꾸는 것이 매우 어려웠다. 유튜브 등 도움이 될 자료들은 모두 코딩에 능숙한 사람들을 대상으로 한 자료가 대부분이었기에 코딩 초보자가 해당 자료를 읽고 완벽하게 모델을 구현하기에는 시간도, 실력도 부족하였다.

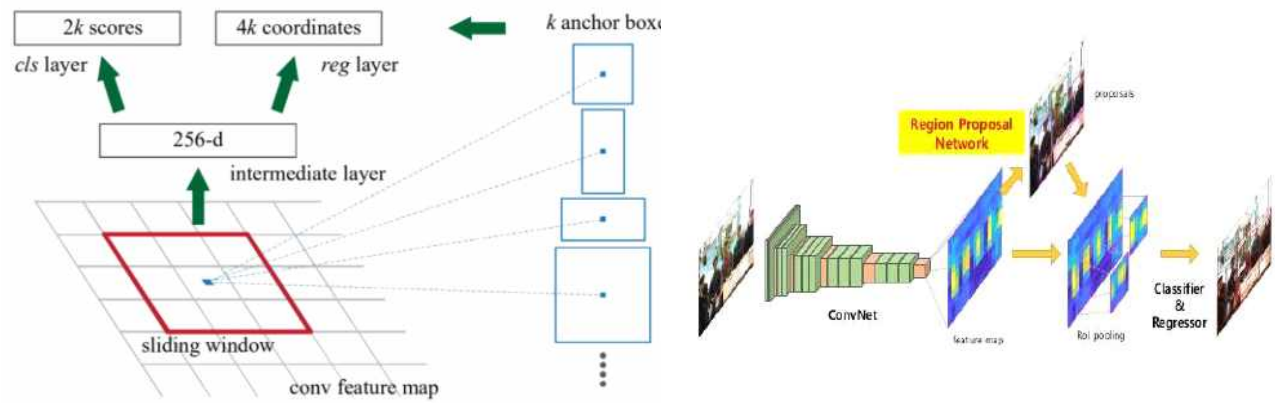
YOLO v3 모델을 추후에라도 제대로 구현하기 위해서는 우선 코딩 실력과 경험을 키워야 한다. 딥러닝 모델 구현에 쓰이는 텐서플로, 파이토치 라이브러리의 기능에 대해 학습하고, 코드를 분석할 수 있도록 프로그래밍 실력도 키워야 한다. 해당 모델의 작동 방식을 더욱 깊게 공부해 구동 방식도 완벽히 이해할 수 있어야한다고 생각한다. 무엇보다도 다양한 프로그램과 라이브러리, 사이트를 사용해 어떤 프로그램을 사용해 코딩하더라도 해당 프로그램을 능숙하게 사용할 수 있어야 한다. 아무리 좋은 프로그램과 라이브러리를 사용하더라도 제대로 사용할 줄 모른다면 해당 프로그램과 라이브러리를 사용하는 이유가 없다. 따라서, 모델을 제대로 구현해 원하는대로 이미지를 학습시키고, 결과를 도출해내기 위해서는 추가적인 공부가 필요하다고 생각한다.

2. Faster RCNN

YOLO v3 구현을 실패하고, 다른 모델을 구글 코랩이 아닌 다른 방법으로 구현하고자 하였다. Faster RCNN 모델을 PyCharm에서 구현하고자 했지만, YOLO v3 모델과 마찬가지로 오류를 해결하지 못해 실패하였다.

2.1 Faster RCNN 이란

Faster R-CNN 방법이란 Fast RCNN 모델이 cpu에서 돌아감으로서 대부분의 시간을 selective search에 소모해 현실에서 사용하기에는 시간이 많이 소모된다는 점을 개선하고자 개발된 모델이다. Fast RCNN은 selective search 과정에서 region proposal을 생성하는 방식을 사용하였지만, Faster RCNN은 region proposal을 생성하는 방법 자체를 CNN 내부에 네트워크(region proposal network,RPN)에 넣어 정확도와 속도를 향상시켰다. RPN이란 feature map이 주어졌을 때, 물체가 있을법한 위치를 예측하는 것이다. 다양한 객체을 인식할 수 있도록 다양한 크기의 k개의 anchor box를 이용한다. 하단 그림을 보면 anchor box의 크기가 다양한 것을 알 수 있다. Feature map 상에서 왼쪽에서 오른쪽으로 움직으며 각 위치마다 intermediate feature vector을 추출하는 sliding window 방식을 이용하며, 이 때 추출된 vector로 regression과 classification을 진행한다. RPN에서의 regression은 bounding box의 위치 (중심 x좌표, 중심 y좌표, 가로 길이, 세로 길이)를 예측해주며, classification은 물체가 있는지 없는지 여부만을 따지는 binary classification이다.



또, 모든 트레이닝을 하나의 Loss 함수로 진행하는 Multi-task Loss를 사용해 학습을 더 빠르고 통합적으로 진행할 수 있도록 하였다. Faster RCNN의 Loss 함수 식을 보면, RPN을 학습하기 위한 Loss와 분류기를 학습하기 위한 Loss를 더한 모습이다.

2.2 구현 방법

구글링을 하여 찾은 코드를 클론 코딩하며 Faster RCNN을 구현하고자 하였다. PyCharm에서 코딩했으며, 코드는 아래와 같다.

```
import torch
import torchvision
from torch.utils.data import Dataset, DataLoader
import os
from PIL import Image
from torchvision import transforms

class CustomDataset(Dataset):    // 데이터 세트를 초기화하고 이미지와 라벨을 읽음
    def __init__(self, root_dir, transforms=None, num_images=25, train=True):
        self.root_dir = root_dir
        self.transforms = transforms
        self.classes = ["car", "person", "dog"]
        self.num_images = num_images
        self.image_paths = []
        self.targets = []
        self.train = train
        for idx, cls in enumerate(self.classes):
            cls_dir = os.path.join(self.root_dir, cls)
            for i in range(1, self.num_images + 1):
                img_file = f"{i}.jpg"
                bbox_file = f"{i}.txt"
                img_path = os.path.join(cls_dir, img_file)
                bbox_path = os.path.join(cls_dir, bbox_file)
                if os.path.exists(img_path):
                    self.image_paths.append(img_path)
                    if self.train and os.path.exists(bbox_path):
                        with open(bbox_path, "r") as file:
                            lines = file.readlines()
                            boxes = []
                            for line in lines:
                                box_info = line.strip().split()
                                x_center, y_center, width, height = map(float, box_info)
                                x_min = x_center - width / 2
                                y_min = y_center - height / 2
                                x_max = x_center + width / 2
                                y_max = y_center + height / 2
                                boxes.append([x_min, y_min, x_max, y_max, idx])
```

```
                            target_dict = {"boxes": torch.tensor(boxes), "labels": torch.tensor([idx] * len(boxes))}
                            self.targets.append(target_dict)
                    elif self.train:
                        print(f"Warning: Bounding box file not found for image: {img_path}")
                    elif not self.train:    # Test 시에는 빈 딕셔너리 반환
                        self.targets.append({})
    def __len__(self):
        return len(self.image_paths)
    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        img = Image.open(img_path).convert("RGB")
        if self.transforms is not None:
            img = self.transforms(img)
        if self.train:
            target = self.targets[idx]
            return img, target
        else:
            return img
transforms = torchvision.transforms.Compose([
    torchvision.transforms.Resize((300, 300)),
    torchvision.transforms.ToTensor(),
])
train_dataset = CustomDataset(root_dir="trainimage", transforms=transforms, num_images=25, train=True)
test_dataset = CustomDataset(root_dir="testimage", transforms=transforms, num_images=10, train=False)
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)
def fasterrcnn_resnet50_fpn_custom(num_classes):
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False)
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = torchvision.models.detection.faster_rcnn.FastRCNNPredictor(in_features, num_classes)
    return model

model = fasterrcnn_resnet50_fpn_custom(num_classes=3)
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
model.to(device)
model.train()

# 학습 루프
num_epochs = 10
optimizer = torch.optim.SGD(model.parameters(), lr=0.005, momentum=0.9, weight_decay=0.0005)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
for epoch in range(num_epochs):
    for images, targets in train_loader:
        optimizer.zero_grad()
        images = list(img.to(device) for img in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets if t]  # targets를 device로 이동
```

```
            # 모델에서 바운딩 박스 예측
            loss_dict = model(images, targets)
            # 학습 코드 유지
            losses = sum(loss for loss in loss_dict.values())
            losses.backward()
            optimizer.step()
        lr_scheduler.step()
# 모델 평가
model.eval()
predictions = []
for images in test_loader:
    images = list(img.to(device) for img in images)
    with torch.no_grad():   # 모델에서 바운딩 박스 예측
        output = model(images)
    predictions.extend(output)   # 출력값을 바운딩 박스로 변환
```

위 코드는 데이터 세트를 초기화하고, 준비한 이미지와 라벨을 읽힌다. 이후 이미지를 전처리하고, Faster RCNN 모델을 불러와 학습시킨다. 이후 테스트할 이미지들을 읽고 예측해 출력한다. 해당 코드를 실행했을 때 계속 targets를 device로 이동시키는 과정에서 오류가 났고, 여러 시도를 해보았지만 끝내 해결하지 못했다.

2.3 실패 사유 분석 및 고찰

YOLO v3 모델의 경우와 마찬가지로 실력과 경험의 부족으로 코드 자체를 완벽히 이해하지 못한 점이 가장 큰 실패 사유라고 생각한다. 어떠한 형태로 Faster RCNN이 작동하는지는 이론 공부를 통해 알 수 있었지만, 이를 코딩으로 직접 구현해내는 것은 다른 이야기였다. GPU를 사용하는 모델이라 실패를 한 건지, 코드를 작성한 사람과 데이터 세트가 다르고, 파일의 위치가 다른 점이 있었던 건지 지식의 부족으로 오류를 해결할 수 없었고 왜 오류가 났는지조차 알 수 없었다.

영상 처리까지는 matlab을 주로 사용하기도 했고, 단계가 명확히 나뉘고 예시 코드도 많아 보다 수월하게 이해하고 실습할 수 있었지만, 딥러닝은 예시 코드들도 프로그래밍 언어 숙달자를 대상으로 하거나, 딥러닝을 배웠던 사람들을 대상으로 한 자료가 많아 이해하기도 상당히 어려웠다. 상당히 많은 시간을 투자했음에도 어떠한 모델도 성공적으로 구현하지 못한 것에 반성하며, 이후 추가적인 학습을 통해 다시 한번 YOLO v3 혹은 Faster RCNN 모델을 구현해보고 싶다. GitHub를 이용해 이미 학습이 완료된 모델들부터 시행해보고 코드를 분석해보며 딥러닝 모델들의 구현은 어떤식으로 코드를 짜야 하는지 알아본 뒤, 다시 한 번 시도해볼 것이다. 간단한 코드를 이용한 영상 처리는 가능하지만, 모델의 구조에 대한 이해와 구현 코드를 이해해야 하는 딥러닝은 추가 학습이 필수적임을 알 수 있었다.

3. 참고 자료

- 꽁소원, 2022.05.14., 05. YOLO 버전별 비교
https://ggongsowon.tistory.com/115

- 김상현, 2021.04.15., Faster R-CNN 논문 리뷰 및 코드 구현
https://velog.io/@skhim520/Faster-R-CNN-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-%EB%B0%8F-%EC%BD%94%EB%93%9C-%EA%B5%AC%ED%98%84

- 로디네로, 2021.04.17., [머신러닝 공부] 딥러닝/Faster RCNN (object detection)
https://dbstndi6316.tistory.com/274

- 박지우, 2022.10.14., [Advanced ML & DL Week3] Faster R-CNN Pytorch 구현
https://kubig-2022-2.tistory.com/79

- 솔라리스, 2018,02,07, 17.텐서플로우(TensorFlow)를 이용한 Faster R-CNN Transfer Learning (Fine-Tuning)으로 나만의 물체 검출기(Object Detector) 만들어보기 - Pet Detector
http://solarisailab.com/archives/2422

- 위키독스, 2-02 YOLOv3 모델 학습하기
https://wikidocs.net/215178

- 초딩Youtube, 2021.04.29., 딥러닝 yolov3 학습파일 만들기
https://www.youtube.com/watch?v=51fZ2FTau7E&list=PLJlV2dBmsof_6J9EWx-CyvpBz7cyWPwhP&index=1

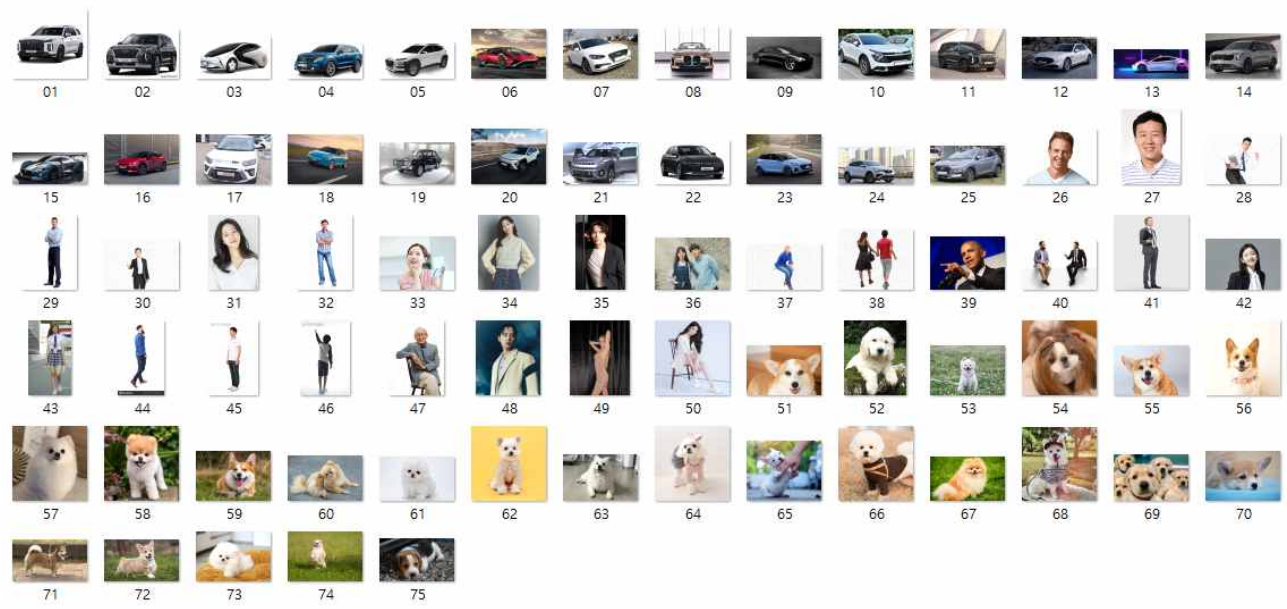
- 초딩Youtube, 2021.04.29., C# yolov3 사용하기(기초편)
https://www.youtube.com/watch?v=33szl_pkSzo

- hannah0125, 2021.11.25., Google Colab을 이용한 YOLOv3 커스텀 데이터 학습하기
https://velog.io/@hannah0125/Google-Colab%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-YOLOv3-%EB%8D%B0%EC%9D%B4%ED%84%B0-%ED%95%99%EC%8A%B5%ED%95%98%EA%B8%B0

- winston1214, 2021.06.21., Object Detection 정리 (RCNN, FastRNN, Faster RCNN 중심으로)
https://bigdata-analyst.tistory.com/269

4. 첨부 자료

4.1 Train images



4.2 Test images